LaxLF: Side Conditions and External Evidence as Monads

Furio Honsell, Luigi Liquori, Ivan Scagnetto

$\mathsf{L}^{\mathsf{ax}}\mathsf{F}$: Side Conditions and External Evidence as Monads

Furio Honsell[1], Luigi Liquori[2], Ivan Scagnetto[1]

[1] Università di Udine, Italy, {*furio.honsell, ivan.scagnetto*}*@uniud.it*
[2] INRIA, Sophia Antipolis Méditerranée, France, *Luigi.Liquori@inria.fr*

Abstract. We extend the constructive dependent type theory of the Logical Framework LF with a family of *monads* indexed by predicates over typed terms. These monads express the *effect* of factoring-out, postponing, or delegating to an *external oracle* the verification of a constraint or a side-condition. This new framework, called *Lax Logical Framework*, $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$, is a conservative extension of LF, and hence it is the appropriate metalanguage for dealing formally with side-conditions or external evidence in logical systems. $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$ is the natural strengthening of $\mathsf{LF}_{\mathcal{P}}$ (the extension of LF introduced by the authors together with Marina Lenisa and Petar Maksimovic), which arises once the *monadic* nature of the *lock* constructors of $\mathsf{LF}_{\mathcal{P}}$ is fully exploited. The nature of these monads allows to utilize the *unlock* destructor instead of Moggi's monadic let_T, thus simplifying the equational theory. The rules for the unlock allow us, furthermore, to remove the monadic constructor once the constraint is satisfied. By way of example we discuss the encodings in $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$ of call-by-value λ-calculus, Hoare's Logic, and Elementary Affine Logic.

1 Introduction

The system $\mathsf{LF}_{\mathcal{P}}$ [18] is a conservative extension of LF. It was introduced to factor out neatly judgements whose justification can be delegated to an *external oracle*. This allows us to recover within a Logical Framework many different *proof cultures* that otherwise can be embedded only very deeply [14] or axiomatically [20]. In particular, recourse in formal proofs to external sources of justification and external evidence such as diagrams, physical analogies, explicit computations according to *Poincaré Principle* [5], and to external proof search tools can thus be explicitly *invoked* and *recorded* in a LF type-theoretic framework. Methodologically this is a simple, but quite significant move, since in dealing with logics one has to rely on external objects more often than one may think. Any *adequacy* result or even the very *execution* of the most obvious rule relies ultimately on some *external* unformalizable convention, as captured by *Münchausen trilemma* [1] or the story of *Achilles and the Tortoise* narrated by Lewis Carroll [7].

The idea behind $\mathsf{LF}_{\mathcal{P}}$ is to express explicitly, by means of a new type constructor $\mathcal{L}^{\mathcal{P}}_{M,\sigma}[\tau]$, that in order to obtain a term of type τ it is necessary to verify the constraint $\mathcal{P}(\Gamma \vdash_{\Sigma} M : \sigma)$. This idea grew out of a series of papers, [6, 17, 19, 18, 15], on extensions of LF published by the authors in recent years.

In this paper we introduce a new system, called *Lax Logical Framework*, $\mathsf{L^{ax}F}$, which amounts to the natural generalization and strengthening of $\mathsf{LF}_{\mathcal{P}}$, once the *monadic* nature of the $\mathcal{L}^{\mathcal{P}}_{M,\sigma}[N]$ constructors is recognized and fully exploited. Hence $\mathsf{L^{ax}F}$ is the extension of LF with a family of *monads* indexed by predicates over typed terms, which capture the *effect* of factoring out and postponing, or delegating to an external oracle the verification of the constraint or side-condition $\mathcal{P}(\Gamma \vdash_{\Sigma} M : \sigma)$.

Our basic idea is that any side condition $\mathcal{P}$ can be viewed as a monad $T_{\mathcal{P}}$, where the categorical natural transformation $\eta_{T_{\mathcal{P}}} : A \to T_{\mathcal{P}}(A)$ expresses the fact that a judgement can always be asserted *weakly*, *i.e.* subject to the satisfaction of a given constraint. While the other natural transformation characterizing a monad $\mu_{T_{\mathcal{P}}} : T^2_{\mathcal{P}}(A) \to T_{\mathcal{P}}(A)$, expresses the fact that it is useless to verify twice a given constraint.

The main extension with respect to the language of $\mathsf{LF}_{\mathcal{P}}$ is that, for $N : \tau$, the destructor $\mathcal{U}^{\mathcal{P}}_{M,\sigma}[N]$ of a particular lock-type, can be used freely provided it is *guarded*, *i.e.* it appears within a subterm whose type has the same lock-type constructor, *i.e.* $\mathcal{L}^{\mathcal{P}}_{M,\sigma}[\rho]$. Thereby, checking predicates in locks can be postponed and, most usefully, functions which output terms of lock-type can be "applied" also to locked-arguments. The nature of these monads allows us to utilize the $\mathcal{U}^{\mathcal{P}}_{M,\sigma}[N]$ destructor instead of the usual monadic let_T, thus greatly simplifying the equational theory. Moreover, as in the case of $\mathsf{LF}_{\mathcal{P}}$, but in addition to what happens with ordinary monads, the rules concerning $\mathcal{U}^{\mathcal{P}}_{M,\sigma}[N]$ allow us to drop the monadic constructor if the constraint is satisfied.

We give classical examples of encodings in $\mathsf{L^{ax}F}$ of logical systems, thereby showing that $\mathsf{L^{ax}F}$ is the appropriate metalanguage for dealing formally with side-conditions, and external evidence. Because of the extra expressive power given by guarded terms of the form $\mathcal{U}^{\mathcal{P}}_{M,\sigma}[N]$, signatures become much more flexible, thus achieving the full modularity that we have been looking for in recent years. We discuss briefly also the intriguing case of Elementary Affine Logic [3].

In conclusion, in this paper we extend:

• the well understood principles of the LF paradigm for explaining a logic, *i.e.* *judgments as types*, and *rules* or *hypothetical judgements as higher order types*, and *schemata as higher order functions*, and *quantified variables as bound metalanguage variables*, with the new clause: *side conditions* and *external evidence as monads*;

• the capacity of logical systems to combine and relate two software tools using a simple communication paradigm via "wrappers".

Related Work. This paper builds on earlier work of the authors [17, 19, 18, 15] and was inspired by the very extensive work on Logical Frameworks by [24, 27, 8, 23, 25, 26]. The term *"Lax"* is borrowed from [9, 21], and indeed our system can be viewed as a generalization, to a family of lax operators, of the work carried out there, as well as Moggi's *partial* λ-calculus [22]. A correspondence between lax modalities and monads in functional programming was pointed out in [2, 12]. In [23, 11, 10] the connection between constraints and monads in logic programming was considered, but to our knowledge this is the first paper which

$$
\begin{array}{rl}
\Sigma \in \mathit{Signatures} & \Sigma ::= \emptyset \mid \Sigma, a{:}K \mid \Sigma, c{:}\sigma \\
K \in \mathit{Kinds} & K ::= \mathsf{Type} \mid \Pi x{:}\sigma.K \\
\sigma, \tau, \rho \in \mathit{Families\ (Types)} & \sigma ::= a \mid \Pi x{:}\sigma.\tau \mid \sigma\ N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \\
M, N \in \mathit{Objects} & M ::= c \mid x \mid \lambda x{:}\sigma.M \mid M\ N \mid \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] \mid \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]
\end{array}
$$

Fig. 1. The pseudo-syntax of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$

$$
(\lambda x{:}\sigma.M)\ N \to_{\beta\mathcal{L}} M[N/x]\ (\beta{\cdot}O{\cdot}Main) \qquad \mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]] \to_{\beta\mathcal{L}} M\ (\mathcal{L}{\cdot}O{\cdot}Main)
$$

Fig. 2. Main one-step-$\beta\mathcal{L}$-reduction rules in $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$

clearly establishes the correspondence between side conditions and monads in a *higher order dependent type theory* and logical frameworks.

Synopsis. In Section 2, we present the syntax and the typing system of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$. In Section 3 we discuss the changes in the metatheory of the framework $\mathsf{LF}_{\mathcal{P}}$, induced by the new typing rule, The conservativity of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$ both w.r.t. $\mathsf{LF}_{\mathcal{P}}$ and to LF is discussed at the end of Section 3. Three case studies are presented in Section 4. Concluding remarks and directions for future work are in Section 5[3].

2 $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$

In this section, we introduce the syntax and the rules of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$: in Figure 1, we give the syntactic categories of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$, namely signatures, contexts, kinds, families (*i.e.*, types) and objects (*i.e.*, terms), while the main one-step $\beta\mathcal{L}$-reduction rules appear in Figure 2. The language of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$ is the same as that of $\mathsf{LF}_{\mathcal{P}}$ [18]. In particular, w.r.t. classical LF, we add the *lock-types* constructor ($\mathcal{L}$) for building types of the shape $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, where $\mathcal{P}$ is a predicate on typed judgements, and correspondingly at object level the constructor lock ($\mathcal{L}$) and destructor unlock ($\mathcal{U}$). The intended meaning of the $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\cdot]$ constructors is that of *logical filters*. Locks can be viewed also as a generalization of the Lax modality of [9, 21]. One of the points of this paper is to show that they can be viewed also as *monads*.

Following the standard specification paradigm of Constructive Type Theory, we define lock-types using introduction, elimination, and equality rules. Namely, we introduce a lock-constructor for building objects $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]$ of type $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]$, via the introduction rule ($O{\cdot}Lock$). Correspondingly, we introduce an unlock-destructor $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[M]$ via the elimination rule ($O{\cdot}Guarded{\cdot}Unlock$). These rules give evidence to the understanding of *locks as monads*[4]. The introduction rule of lock-types immediately corresponds to the introduction rule of monads, but this is not so immediate for the elimination rule which is normally given for monads

[3] A web appendix is available (for interested readers) at http://www.dimi.uniud.it/scagnett/pubs/MonadixLFP-Appendix.pdf

[4] Given a predicate $\mathcal{P}$ and $\Gamma \vdash_{\Sigma} N : \sigma$, the intended monad $(T_{\mathcal{P}}, \eta, \mu)$ can be naturally defined on the term model of $\mathsf{L}^{\mathsf{ax}}\mathsf{F}$ viewed as a category. In particular $\eta_{\rho} \triangleq \lambda x{:}\rho.\mathcal{L}^{\mathcal{P}}_{N,\sigma}[x]$ and $\mu_{\rho} \triangleq \lambda x{:}\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]].\ \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\mathcal{U}^{\mathcal{P}}_{N,\sigma}[x]]$.

Signature rules

$$\frac{}{\emptyset \text{ sig}}\ (S{\cdot}Empty)$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}}\ (S{\cdot}Kind)$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma \sigma{:}\mathsf{Type} \quad c \notin \mathsf{Dom}(\Sigma)}{\Sigma, c{:}\sigma \text{ sig}}\ (S{\cdot}Type)$$

Context rules

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \emptyset}\ (C{\cdot}Empty)$$

$$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma{:}\mathsf{Type} \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (C{\cdot}Type)$$

Kind rules

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}}\ (K{\cdot}Type)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.K}\ (K{\cdot}Pi)$$

Family rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\ (F{\cdot}Const)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}\sigma.\tau : \mathsf{Type}}\ (F{\cdot}Pi)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi x{:}\tau.K \quad \Gamma \vdash_\Sigma N : \tau}{\Gamma \vdash_\Sigma \sigma\, N : K[N/x]}\ (F{\cdot}App)$$

$$\frac{\Gamma \vdash_\Sigma \rho : \mathsf{Type} \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] : \mathsf{Type}}\ (F{\cdot}Lock)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : K \quad \Gamma \vdash_\Sigma K' \quad K{=}_{\beta\mathcal{L}}K'}{\Gamma \vdash_\Sigma \sigma : K'}\ (F{\cdot}Conv)$$

Object rules

$$\frac{\vdash_\Sigma \Gamma \quad c{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma c : \sigma}\ (O{\cdot}Const)$$

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (O{\cdot}Var)$$

$$\frac{\Gamma, x{:}\sigma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda x{:}\sigma.M : \Pi x{:}\sigma.\tau}\ (O{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\, N : \tau[N/x]}\ (O{\cdot}App)$$

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type} \quad \sigma{=}_{\beta\mathcal{L}}\tau}{\Gamma \vdash_\Sigma M : \tau}\ (O{\cdot}Conv)$$

$$\frac{\Gamma \vdash_\Sigma M : \rho \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma \mathcal{L}^{\mathcal{P}}_{N,\sigma}[M] : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho]}\ (O{\cdot}Lock)$$

$$\frac{\Gamma \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{N,\sigma}[\rho] \quad \mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)}{\Gamma \vdash_\Sigma \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \rho}\ (O{\cdot}Top{\cdot}Unlock)$$

$$\frac{\Gamma, x{:}\tau \vdash_\Sigma M : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho] \quad \Gamma \vdash_\Sigma N : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau]}{\Gamma \vdash_\Sigma M[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x] : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x]]}\ (O{\cdot}Guarded{\cdot}Unlock)$$

Fig. 3. The L$^{\text{a}\times}$F Type System

using a let_T-construct. The correspondence becomes clear once we realize that $let_{T_{\mathcal{P}(\Gamma \vdash S:\sigma)}} x = M \; in \; N$ can be safely replaced by $N[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[M]/x]$ since the $\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\cdot]$-monads satisfy the property $let_{T_{\mathcal{P}}}\ x = M \; in \; N \to N$ if $x \notin FV(N)$, provided x occurs *guarded* in N, *i.e.* within subterms of the appropriate locked-type.

Finally, to recover the intended meaning of $\mathcal{L}^{\mathcal{P}}_{N,\sigma}[\cdot]$, we need to introduce in L$^{\text{a}\times}$F also $(O{\cdot}Top{\cdot}Unlock)$, which allows for the elimination of the lock-type constructor if the predicate $\mathcal{P}$ is verified, possibly *externally*, on an appropriate and derivable judgement. Figure 3 shows the full typing system of L$^{\text{a}\times}$F. The type equality rule of L$^{\text{a}\times}$F uses a notion of conversion which derives from $\beta\mathcal{L}$-reduction, a combination of standard β-reduction, $(\beta{\cdot}O{\cdot}Main)$, with another notion of

reduction ($\mathcal{L}{\cdot}O{\cdot}Main$), called $\mathcal{L}$-reduction. The latter behaves as a lock-releasing mechanism, erasing the $\mathcal{U}$-$\mathcal{L}$ pair in a term of the form $\mathcal{U}^{\mathcal{P}}_{N,\sigma}[\mathcal{L}^{\mathcal{P}}_{N,\sigma}[M]]$.

Since external predicates affect reductions in $\mathsf{L^{ax}F}$, they must be well-behaved in order to preserve subject reduction. And this property is needed for decidability, possibly *up to an oracle*, which is essential in LF's.

Definition 1 (Well-behaved predicates, [18]). *A finite set of predicates* $\{\mathcal{P}_i\}_{i\in I}$ *is* well-behaved *if each* $\mathcal{P}$ *in the set satisfies the following conditions:*

- ***Closure under signature and context weakening and permutation:***
  1. *If* Σ *and* Ω *are valid signatures such that* $\Sigma \subseteq \Omega$ *and* $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$*, then* $\mathcal{P}(\Gamma \vdash_\Omega \alpha)$.
  2. *If* Γ *and* Δ *are valid contexts such that* $\Gamma \subseteq \Delta$ *and* $\mathcal{P}(\Gamma \vdash_\Sigma \alpha)$*, then* $\mathcal{P}(\Delta \vdash_\Sigma \alpha)$.
- ***Closure under substitution:*** *If* $\mathcal{P}(\Gamma, x{:}\sigma', \Gamma' \vdash_\Sigma N : \sigma)$ *and* $\Gamma \vdash_\Sigma N' : \sigma'$*, then* $\mathcal{P}(\Gamma, \Gamma'[N'/x] \vdash_\Sigma N[N'/x] : \sigma[N'/x])$.
- ***Closure under reduction:***
  1. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *and* $N \to_{\beta\mathcal{L}} N'$*, then* $\mathcal{P}(\Gamma \vdash_\Sigma N' : \sigma)$.
  2. *If* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma)$ *and* $\sigma \to_{\beta\mathcal{L}} \sigma'$*, then* $\mathcal{P}(\Gamma \vdash_\Sigma N : \sigma')$.

3 Metatheory of $\mathsf{L^{ax}F}$

The proofs of the metatheoretic properties of $\mathsf{L^{ax}F}$ follow the pattern of [18].

Strong Normalisation and Confluence
The proof of strong normalization relies on that of LF [13]. First, we introduce the function $\cdot^{-\mathcal{UL}} : \mathsf{L^{ax}F} \to \mathsf{LF}$, mapping $\mathsf{L^{ax}F}$ terms into LF terms by deleting the $\mathcal{L}$ and $\mathcal{U}$ symbols[5]. The proof then proceeds by contradiction, assuming a term T with an infinite $\beta\mathcal{L}$-reduction sequence. Next, we prove that only a finite number of β-reductions can be performed within any given $\mathsf{L^{ax}F}$ term T. Whence, in order for T to have an infinite $\beta\mathcal{L}$-reduction sequence, it must have an infinite $\mathcal{L}$-sequence, which is impossible, obtaining the contradiction. We highlight only the crucial case of the rule ($O{\cdot}Guarded{\cdot}Unlock$). Its conclusion is translated to LF as follows:

$$\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}}[(\lambda x_f{:}\sigma^{-\mathcal{UL}}.N^{-\mathcal{UL}})\, S^{-\mathcal{UL}}/x] : (\lambda y_f{:}\sigma^{-\mathcal{UL}}.(\rho[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x])^{-\mathcal{UL}})\, S^{-\mathcal{UL}}$$

The latter judgment, through standard β-reduction yields:

$$\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x] : \rho^{-\mathcal{UL}}[(\lambda x_f{:}\sigma^{-\mathcal{UL}}.N^{-\mathcal{UL}})\, S^{-\mathcal{UL}}/x],$$

i.e. $\Gamma^{-\mathcal{UL}} \vdash_{\Sigma^{-\mathcal{UL}}} M^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x] : \rho^{-\mathcal{UL}}[N^{-\mathcal{UL}}/x]$. Thus, only a finite number of β-reductions can be performed within the translation of any given $\mathsf{L^{ax}F}$ term T and we can proceed by contradiction. Thus we have:

[5] $.^{-\mathcal{UL}}$ is the identity over constants and variables and it is recursively applied to subterms of Π,λ and application constructors, preserving their structure. The only interesting cases are those involving the $\mathcal{L}$ and $\mathcal{U}$ constructors: $(\mathcal{L}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} \triangleq (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$ and $(\mathcal{U}^{\mathcal{P}}_{N,\sigma}[T])^{-\mathcal{UL}} \triangleq (\lambda x_f{:}\sigma^{-\mathcal{UL}}.T^{-\mathcal{UL}})\, N^{-\mathcal{UL}}$, where x_f is a variable which *does not* occur free in T. Its purpose is to preserve the N and σ in the subscript of the $\mathcal{L}$ and $\mathcal{U}$ symbols, while being able to β-reduce to T in one step.

Theorem 1 (Strong Normalization of $\mathsf{L}^{a\times}\mathsf{F}$).
1. If $\Gamma \vdash_\Sigma K$, then K is $\beta\mathcal{L}$-strongly normalizing.
2. if $\Gamma \vdash_\Sigma \sigma : K$, then σ is $\beta\mathcal{L}$-strongly normalizing.
3. if $\Gamma \vdash_\Sigma M : \sigma$, then M is $\beta\mathcal{L}$-strongly normalizing.

Confluence is proved as for $\mathsf{LF}_\mathcal{P}$, using *Newman's Lemma* ([4], Chapter 3), and showing that the reduction on "raw terms" is locally confluent. Hence, we have:

Theorem 2 (Confluence of $\mathsf{L}^{a\times}\mathsf{F}$). *$\beta\mathcal{L}$-reduction is confluent, i.e.:*
1. If $K \twoheadrightarrow_{\beta\mathcal{L}} K'$ and $K \twoheadrightarrow_{\beta\mathcal{L}} K''$, then there exists a K''' such that $K' \twoheadrightarrow_{\beta\mathcal{L}} K'''$ and $K'' \twoheadrightarrow_{\beta\mathcal{L}} K'''$.
2. If $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma'$ and $\sigma \twoheadrightarrow_{\beta\mathcal{L}} \sigma''$, then there exists a σ''' such that $\sigma' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$ and $\sigma'' \twoheadrightarrow_{\beta\mathcal{L}} \sigma'''$.
3. If $M \twoheadrightarrow_{\beta\mathcal{L}} M'$ and $M \twoheadrightarrow_{\beta\mathcal{L}} M''$, then there exists an M''' such that $M' \twoheadrightarrow_{\beta\mathcal{L}} M'''$ and $M'' \twoheadrightarrow_{\beta\mathcal{L}} M'''$.

Subject Reduction
Inversion and subderivation properties play a key role in the proof of subject reduction (SR). However, in $\mathsf{L}^{a\times}\mathsf{F}$, given a derivation of $\Gamma \vdash_\Sigma \alpha$ and a subterm N occurring in the subject of this judgement, we cannot prove that there always exists a derivation of the form $\Gamma \vdash_\Sigma N : \tau$ (for a suitable τ). Consider the following example. Clearly $\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^\mathcal{P}_{S,\sigma}[x] : \mathcal{L}^\mathcal{P}_{S,\sigma}[\tau]$ holds, and assume that $\Gamma \vdash_\Sigma N : \mathcal{L}^\mathcal{P}_{S,\sigma}[\tau]$; we then have

$$\frac{\Gamma, x{:}\tau \vdash_\Sigma \mathcal{L}^\mathcal{P}_{S,\sigma}[x] : \mathcal{L}^\mathcal{P}_{S,\sigma}[\tau] \quad \Gamma \vdash_\Sigma N : \mathcal{L}^\mathcal{P}_{S,\sigma}[\tau]}{\Gamma \vdash_\Sigma \mathcal{L}^\mathcal{P}_{S,\sigma}[\mathcal{U}^\mathcal{P}_{S,\sigma}[N]] : \mathcal{L}^\mathcal{P}_{S,\sigma}[\tau]}\ (O{\cdot}Guarded{\cdot}Unlock)$$

but if $\mathcal{P}(\Gamma \vdash_\Sigma S : \sigma)$ does not hold, and τ is not a lock-type, then we cannot derive any judgement whose subject is $\mathcal{U}^\mathcal{P}_{S,\sigma}[N] \ : \ \tau$. Hence we have to restate point 6 of Proposition 3.11 (Subderivation, part 1) of [18] as follows:

Proposition 1 (Subderivation, part 1, point 6). *Given a derivation $\mathcal{D}$ of the judgement $\Gamma \vdash_\Sigma \alpha$, and a subterm N occurring in the subject of this judgement, we have that either there exists a derivation of a judgement having N as a subject, or there exists a derivation of a judgment having N' as a subject, where $N \equiv \mathcal{U}^\mathcal{P}_{S,\sigma}[N']$ (for suitable $\mathcal{P}$, S and σ).*

The proof is straightforward. And straightforward is also the extension to $\mathsf{L}^{a\times}\mathsf{F}$ of the rest of the proof of SR for $\mathsf{LF}_\mathcal{P}$ in [18]. Thus we establish the fundamental:

Theorem 3 (Subject Reduction of $\mathsf{L}^{a\times}\mathsf{F}$). *If predicates are well-behaved, then:*
1. If $\Gamma \vdash_\Sigma K$ and $K \to_{\beta\mathcal{L}} K'$, then $\Gamma \vdash_\Sigma K'$.
2. If $\Gamma \vdash_\Sigma \sigma : K$ and $\sigma \to_{\beta\mathcal{L}} \sigma'$, then $\Gamma \vdash_\Sigma \sigma' : K$.
3. If $\Gamma \vdash_\Sigma M : \sigma$ and $M \to_{\beta\mathcal{L}} M'$, then $\Gamma \vdash_\Sigma M' : \sigma$.

The issue of decidability for $\mathsf{L}^{a\times}\mathsf{F}$ can be addressed as that for $\mathsf{LF}_\mathcal{P}$ in [18].

Conservativity
We recall that a system $\mathcal{S}'$ is a conservative extension of $\mathcal{S}$ if the language of $\mathcal{S}$ is included in that of $\mathcal{S}'$, and moreover for all judgements $\mathcal{J}$, in the language of $\mathcal{S}$, $\mathcal{J}$ is provable in $\mathcal{S}'$ if and only if $\mathcal{J}$ is provable in $\mathcal{S}$.

Theorem 4 (Conservativity of $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$). $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ *is a conservative extension of* LF.
Proof. (sketch) The *if* part is trivial. For the *only if* part, consider a derivation in $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ and drop all locks/unlocks (*i.e.* release the terms and types originally locked). This pruned derivation is a legal derivation in standard LF.

Notice that the above result holds independently of the particular nature or properties of the external oracles that we may *invoke* during the proof development (in $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$), *e.g.* decidability or recursive enumerability of $\mathcal{P}$.

Instead, $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ is *not* a conservative extension of $\mathsf{LF}_{\mathcal{P}}$, since the new typing rule allows us to derive more judgements with unlocked-subject even if the predicate does not hold *e.g.*

$$\frac{\Gamma, x{:}\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] \vdash_{\Sigma} x : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau] \quad \Gamma \vdash_{\Sigma} N : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau]]}{\Gamma \vdash_{\Sigma} x[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x] : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x]]}\ (O{\cdot}Guarded{\cdot}Unlock)$$

Then, since x does not occur free in τ, $\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau[\mathcal{U}^{\mathcal{P}}_{S,\sigma}[N]/x]] \equiv \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau]$ and we get $\Gamma \vdash_{\Sigma} \mathcal{U}^{\mathcal{P}}_{S,\sigma}[N] : \mathcal{L}^{\mathcal{P}}_{S,\sigma}[\tau]$. We close this Section on $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ with a sort of "hygiene" theorem for the unguarded $\mathcal{U}$-destructor:

Theorem 5 (Soundness of unlock). *If* $\Gamma \vdash_{\Sigma} \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \tau$ *is derived in* $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ *and* Γ *does not contain variables ranging over lock-types (i.e.,* $x{:}\mathcal{L}^{\mathcal{P}}_{S,\sigma}[\rho] \notin \Gamma$*), then* $\mathcal{P}(\Gamma \vdash_{\Sigma} N : \sigma)$ *is true.*
Proof. The proof can be carried out by a straightforward induction on the derivation of $\Gamma \vdash_{\Sigma} \mathcal{U}^{\mathcal{P}}_{N,\sigma}[M] : \tau$.

4 Case Studies

In this section we discuss encodings of logics in $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$. Of course, all encodings given in [18] for $\mathsf{LF}_{\mathcal{P}}$, carry over immediately to the setting of $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$, because the latter is a language extension of the former. So here, we do not present encodings for *modal* and *ordered linear logic*. However, the possibility of using guarded unlocks, *i.e.* the full power of the monad destructor, allows for significant simplifications in several of the encodings of logical systems given in $\mathsf{LF}_{\mathcal{P}}$. We illustrate this point discussing call-by-value λ_v-calculus, which greatly benefits from the possibility of applying functions to locked-arguments, and Hoare's Logic, which combines various kinds of syntactical and semantical locks in its rules. We do not discuss adequacy of these encodings since it is a trivial variant of the one presented in [18]. Finally we discuss a very subtle natural deduction logic, *i.e.* Elementary Affine Logic. That encoding will illustrate how locks can be used to deal with *pattern matching*, and *terms rewriting*, and open up the road to embedding *logic programming* in type theory.

Call-by-value λ_v-calculus
We encode, using *Higher Order Abstract Syntax* (HOAS), the syntax of untyped λ-calculus: $M, N ::= x \mid M\ N \mid \lambda x.M$ as in [18], where natural numbers (through the constructor `free`) are used to represent free variables, while bound variables are rendered as metavariables of $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ of type `term`:

Definition 2 ($\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ signature Σ_λ for untyped λ-calculus).

```
term : Type                        nat  :  Type                        O : nat
   S :  nat -> nat                 free :   nat -> term
 app : term -> term -> term        lam  : (term -> term) -> term
```

Definition 3 (Call-by-value reduction strategy). *The call-by-value (CBV) evaluation strategy is given by:*

$$\frac{}{\vdash_{CBV} M = M}\ (refl) \qquad \frac{\vdash_{CBV} N = M}{\vdash_{CBV} M = N}\ (symm)$$

$$\frac{\vdash_{CBV} M = N \quad \vdash_{CBV} N = P}{\vdash_{CBV} M = P}\ (trans) \qquad \frac{\vdash_{CBV} M = N \quad \vdash_{CBV} M' = N'}{\vdash_{CBV} M\,M' = N\,N'}\ (app)$$

$$\frac{v \text{ is a value}}{\vdash_{CBV} (\lambda x.M)\,v = M[v/x]}\ (\beta_v) \qquad \frac{\vdash_{CBV} M = N}{\vdash_{CBV} \lambda x.M = \lambda x.N}\ (\xi_v)$$

where values are either variables, constants, or abstractions.

The new typing rule ($O{\cdot}Guarded{\cdot}Unlock$) of LaxF, allows to encode naturally the system as follows.

Definition 4 (LaxF signature Σ_{CBV} for λ-calculus CBV reduction). *We extend the signature of Definition 2 as follows:*

```
   eq : term->term->Type
 refl : Π M:term.                (eq M M)
 symm : Π M:term. Π N:term.      (eq N M)->(eq M N)
trans : Π M,N,P:term.            (eq M N)->(eq N P) ->(eq M P)
eq_app: Π M,N,M',N':term.        (eq M N)->(eq M'N')->(eq (app M M')(app N N'))
betav : Π M:  (term->term). Π N:term. L^Val_{N,term}[(eq (app (lam M) N)(M N))]
 csiv : Π M,N:(term->term).(Π x:term. L^Val_{x,term}[(eq (M x)(N x))])->(eq (lam M)(lam N))
```

where the predicate `Val` *is defined as follows:* `Val` $(\Gamma \vdash_\Sigma$ `N : term`$)$ *holds iff either* `N` *is an abstraction or a constant (i.e. a term of the shape* `(free i)`*).*

Notice the neat improvement w.r.t. to the encoding of $\mathsf{LF}_{\mathcal{P}}$, given in [18], as far as the rule `csiv`. The encoding of the rule ξ_{v} is problematic if bound variables are encoded using metavariables, because the predicate `Val` appearing in the lock cannot mention explicitly variables, for it to be well-behaved. In [18], since we could not apply the rules unless we had explicitly eliminated the `Val`-lock, in order to overcome the difficulty we had to make a detour using constants. In LaxF, on the other hand, we can apply the rules "under `Val`", so to speak, and postpone the proof of the `Val`-checks till the very end, and then rather than checking `Val` we can get rid of the lock altogether, since the bound variable of the rule `csiv`, is allowed to be locked. Notice that this phrasing of the rule `csiv` amounts precisely to the fact that in λ_v variables range over values. As a concrete example of all this, we show how to derive the equation $\lambda x.z\,((\lambda y.y)\,x) = \lambda x.z\,x$. Using "pencil and paper" we would proceed as follows:

$$\dfrac{\dfrac{\dfrac{-}{\vdash_{CBV} z = z}\ (refl) \qquad \dfrac{x \text{ is a value}}{(\lambda y.y)\,x = y[x/y]}\ (\beta_v)}{\vdash_{CBV} z\,((\lambda y.y)\,x) = z\,x}\ (app)}{\vdash_{CBV} \lambda x.z\,((\lambda y.y)\,x) = \lambda x.z\,x}\ (\xi_v)$$

Similarly, in $\mathsf{L^{a\times}F}$, we can derive $\mathtt{z{:}term} \vdash_{\Sigma} (\mathtt{refl\ z}) : (\mathtt{eq\ z\ z})$ and
$\Gamma, \mathtt{x{:}term} \vdash_{\Sigma} (\mathtt{betav}\,(\lambda \mathtt{y{:}term.y})\,\mathtt{x}) : \mathcal{L}^{\mathit{Val}}_{\mathtt{x,term}}[(\mathtt{eq}\,(\mathtt{app}\,(\mathtt{lam}\,\lambda \mathtt{y{:}term.y})\,\mathtt{x})\,((\lambda \mathtt{y{:}term.y})\,\mathtt{x}))]$.

This far, in old $\mathsf{LF}_{\mathcal{P}}$, we would be blocked if we could not prove that $\mathit{Val}\,(\Gamma, \mathtt{x{:}term} \vdash_{\Sigma} \mathtt{x : term})$ holds, since `eq_app` cannot accept an argument with a locked-type. However, in $\mathsf{L^{a\times}F}$, we can apply the $(O{\cdot}Guarded{\cdot}Unlock)$ rule obtaining the following proof term (from the typing environment $\Gamma, \mathtt{x{:}term}, \mathtt{z{:}term}$):

$$(\mathtt{eq\_app\ z\ z}\ (\mathtt{app}\ (\mathtt{lam}\ \lambda \mathtt{y{:}term.y})\ \mathtt{x})\ \mathtt{x}\ (\mathtt{refl\ z})\ \mathcal{U}^{\mathit{Val}}_{\mathtt{x,term}}[(\mathtt{betav}\ (\lambda \mathtt{y{:}term.y})\ \mathtt{x})]),$$

of type $\mathcal{L}^{\mathit{Val}}_{\mathtt{x,term}}[(\mathtt{eq}\ (\mathtt{app\ z}\ (\mathtt{app}\ (\mathtt{lam}\ \lambda \mathtt{y{:}term.y})\ \mathtt{x}))\ (\mathtt{app\ z\ x}))]$. And abstracting `x`, a direct application of `csiv` yields the result.

Imp with Hoare Logic

An area of Logic which can greatly benefit from the new system $\mathsf{L^{a\times}F}$ is *program logics*, because of the many syntactical checks which occur in these systems. For lack of space we can discuss only a few rules of Hoare's Logic for a very simple imperative language Imp, whose syntax is:

$p ::= \mathit{skip} \mid x := \mathit{expr} \mid p; p \mid$	null | assignment | sequence
$\mathit{if\ cond\ then\ p\ else\ p} \mid \mathit{while\ cond}\ \{p\}$	cond | while

In [18] we presented an encoding of Hoare's logic for Imp in $\mathsf{LF}_{\mathcal{P}}$ which delegated to external predicates the tedious and subtle checks that boolean expressions, in the *if* and *while* commands, are *quantifier free* (QF predicate) as well as the *non-interference* property in the assignment command. These syntactic constraints on the conditional and loop commands were rendered in $\mathsf{LF}_{\mathcal{P}}$ as follows:

```
bool,prog : Type
Iif       : Πe:bool.prog -> prog ->L^QF_{e,bool}[prog]
Iwhile    : Πe:bool.prog -> L^QF_{e,bool}[prog]
```

where the predicate $\mathit{QF}(\Gamma \vdash_{\Sigma_{Imp}} \mathtt{e : bool})$ holds iff the formula `e` is *closed* and quantifier-free, *i.e.*, it does not contain the `forall` constructor. We can look at QF as a "good formation" predicate, filtering out *bad programs with invalid boolean expressions* by means of "stuck" terms. Thus, the encoding function $\epsilon^{\mathtt{prog}}_{\mathcal{X}}$ mapping programs of the source language Imp, with free variables in $\mathcal{X}$, to the corresponding terms of $\mathsf{LF}_{\mathcal{P}}$ could be defined very easily as follows[6]:

$\epsilon^{\mathtt{prog}}_{\mathcal{X}}(\mathit{if}\ e\ \mathit{then}\ p\ \mathit{else}\ p') = \mathcal{U}^{\mathit{QF}}_{\epsilon^{\mathtt{exp}}_{\mathcal{X}}(e),\mathtt{bool}}[(\mathtt{Iif}\ \epsilon^{\mathtt{exp}}_{\mathcal{X}}(e)\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(p)\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(p'))]$ (*)

$\epsilon^{\mathtt{prog}}_{\mathcal{X}}(\mathit{while}\ e\ \{p\}) = \mathcal{U}^{\mathit{QF}}_{\epsilon^{\mathtt{exp}}_{\mathcal{X}}(e),\mathtt{bool}}[(\mathtt{Iwhile}\ \epsilon^{\mathtt{exp}}_{\mathcal{X}}(e)\ \epsilon^{\mathtt{prog}}_{\mathcal{X}}(p))]$ (*)

(*) if e is a quantifier-free formula. However the terms on the right hand side cannot be directly expressed in $\mathsf{LF}_{\mathcal{P}}$ because if $\mathit{QF}(\Gamma \vdash_{\Sigma_{Imp}} \epsilon^{\mathtt{exp}}_{\mathcal{X}}(e) : \mathtt{bool})$ does not hold, we cannot use the unlock operator. Thus we could be left with two terms of type $\mathcal{L}^{\mathit{QF}}_{\epsilon^{\mathtt{exp}}_{\mathcal{X}}(e),\mathtt{bool}}[\mathtt{prog}]$, instead of type `prog`. This is precisely the limit of the $\mathsf{LF}_{\mathcal{P}}$ encoding in [18]. Since a $\mathcal{U}$-term can only be introduced if the corresponding predicate holds, when we represent rules of Hoare Logic we are forced to consider only legal terms, and this ultimately amounts to restricting explicitly the object language in a way such that QF always returns true.

In $\mathsf{L^{a\times}F}$, instead, we can use naturally the following signature for representing Hoare's Logic, without assuming anything about the object language terms:

[6] For lack of space, we report only the cases of the conditional/loop commands.

$$\texttt{hoare : bool -> prog -> bool -> Type}$$

$$\texttt{hoare\_Iif} : \Pi \texttt{e,e',b:bool}.\Pi \texttt{p,p':prog}.\texttt{(hoare (b and e) p e') ->}\ \texttt{(hoare ((not b) and e) p' e') ->}\ \mathcal{L}^{QF}_{\texttt{b,bool}}[\texttt{(hoare e}\ \mathcal{U}^{QF}_{\texttt{b,bool}}[\texttt{(Iif b p p')] e')}]$$

$$\texttt{hoare\_Iwhile} : \Pi \texttt{e,b:bool}.\Pi \texttt{p:prog}.\texttt{(hoare (e and b) p e) ->}\ \mathcal{L}^{QF}_{\texttt{b,bool}}[\texttt{(hoare e}\ \mathcal{U}^{QF}_{\texttt{b,bool}}[\texttt{(Iwhile b p)] ((not b) and e))}]$$

Moreover, the (*O·Guarded·Unlock*) rule allows also to "postpone" the verification that $\mathit{QF}(\Gamma \vdash_{\Sigma} \texttt{e : bool})$ holds (*i.e.*, that the formula `e` is quantifier-free).

Elementary Affine Logic

We provide a *shallow* encoding of *Elementary Affine Logic* as presented in [3]. This example will exemplify how locks can be used to deal with syntactical manipulations as in the promotion rule of Elementary Affine Logic, which clearly introduces a recursive processing of the context.

Definition 5 (Elementary Affine Logic). *Elementary Affine Logic can be specified by the following rules:*

$$\frac{}{A \vdash_{\mathrm{EAL}} A}\ (Var) \qquad \frac{\Gamma \vdash_{\mathrm{EAL}} B}{\Gamma, A \vdash_{\mathrm{EAL}} B}\ (Weak) \qquad \frac{\Gamma, A \vdash_{\mathrm{EAL}} B}{\Gamma \vdash_{\mathrm{EAL}} A \multimap B}\ (Abst)$$

$$\frac{\Gamma \vdash_{\mathrm{EAL}} A \quad \Delta \vdash_{\mathrm{EAL}} A \multimap B}{\Gamma, \Delta \vdash_{\mathrm{EAL}} B}\ (Appl) \qquad \frac{\Gamma \vdash_{\mathrm{EAL}} !A \quad \Delta, !A, \ldots, !A \vdash_{\mathrm{EAL}} B}{\Gamma, \Delta \vdash_{\mathrm{EAL}} B}\ (Contr)$$

$$\frac{A_1, \ldots, A_n \vdash_{\mathrm{EAL}} A \quad \Gamma_1 \vdash_{\mathrm{EAL}} !A_1 \quad \ldots \quad \Gamma_n \vdash_{\mathrm{EAL}} !A_n}{\Gamma_1 \ldots \Gamma_n \vdash_{\mathrm{EAL}} !A}\ (Prom)$$

Definition 6 (LaxF signature Σ_{EAL} for Elementary Affine Logic).

$$\texttt{o : Type} \qquad \texttt{T : o -> Type} \qquad \texttt{! : o -> o}$$

$$\texttt{c\_appl} : \Pi \texttt{A,B :o. T(A) -> T(A} \multimap \texttt{B) -> T(B)}$$

$$\texttt{c\_abstr} : \Pi \texttt{A,B :o.}\ \Pi \texttt{x:(T(A) -> T(B)).}\ \mathcal{L}^{\mathit{Light}}_{\texttt{x,T(A)->T(B)}}[\texttt{T(A} \multimap \texttt{B)}]$$

$$\texttt{c\_prom} : \Pi \texttt{A,A':o.}\ \Pi \texttt{x: T(A).}\mathcal{L}^{\mathit{Closed}}_{\texttt{x,T(A)}}[\mathcal{L}^{\mathit{Prom}}_{\texttt{<x,A,A'>,T(A)XoXo}}[\texttt{T(A')}]]$$

where `o` *is the type of propositions,* $\multimap$ *and* `!` *are the obvious syntactic constructors,* `T` *is the basic judgement, and* $< \texttt{x}, \texttt{y}, \texttt{z} >$ *denotes any encoding of triples, whose type is denoted by* $\mu\texttt{X}\sigma\texttt{X}\tau$*, e.g.* $\lambda\texttt{u}{:}\mu \to \sigma \to \tau \to \rho.\ \texttt{u x y z} : (\mu \to \sigma \to \tau \to \rho) \to \rho$*. The predicates involved in the locks are defined as follows:*

- *Light* $(\Gamma \vdash_{\Sigma_{\mathrm{EAL}}} \texttt{x} : \texttt{T(A)} \to \texttt{T(B)})$ *holds iff if* `A` *is not of the shape* `!A` *then the bound variable of* `x` *occurs at most once in the normal form of* `x`*.*
- *Closed* $(\Gamma \vdash_{\Sigma_{\mathrm{EAL}}} \texttt{x} : \texttt{T(A)})$ *holds iff there are no free variables of type* `T(B)`*, for some* `B : o` *in* `x`*.*
- *Prom* $(\Gamma \vdash_{\Sigma_{\mathrm{EAL}}} < \texttt{x}, \texttt{A}, \texttt{A}' > : \texttt{T(A)XoXo})$ *holds iff* $\texttt{A} \equiv (\texttt{A}_1 \multimap \texttt{A}_2 \multimap \ldots \multimap \texttt{A}_n)$ *and* $\texttt{A}' \equiv (!\texttt{A}_1 \multimap !\texttt{A}_2 \multimap \ldots \multimap !\texttt{A}_n)$ *and* $\texttt{A}_1, \texttt{A}_2, \ldots, \texttt{A}_n$ *are the arguments of the* `c_abstr`*-constructors in the derivation of* `x`*.*

A few remarks are mandatory. The promotion rule in [3] is in fact a family of natural deduction rules with an arbitrary number of assumptions. Our encoding achieves this via a number of application-rules. Adequacy for this signature can be achieved only in the general formulation of [18], namely:

Theorem 6 (Adequacy for Elementary Affine Logic). $A_1 \dots A_n \vdash_{\mathrm{EAL}} A$ *iff there exists* $\mathtt{M}$ *and* $\mathtt{A_1}{:}\mathtt{o}, \dots, \mathtt{A_n}{:}\mathtt{o}, \mathtt{x_1}{:}\mathtt{T}(\mathtt{A_1}), \dots, \mathtt{x_n}{:}\mathtt{T}(\mathtt{A_n}) \vdash_{\Sigma_{\mathrm{EAL}}} \mathtt{M} : \mathtt{T}(\mathtt{A})$ *and all variables* $\mathtt{x_i}$ *(*$\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$*) occurring more than once in* $\mathtt{M}$ *have type of the shape* $\mathtt{T}(!\mathtt{A_i})$.

The check on the context of the Adequacy Theorem is *external* to the system $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$, but this is in the nature of results which relate *internal* and *external* concepts. *E.g.* the very concept of $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ context, which appears in any Adequacy result, is external to $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$. This check is internalized if the term is closed.

5 Concluding Remarks

In this paper we have shown how to extend LF with a class of monads which capture the effect of delegating to an external oracle the task of providing part of the necessary evidence for establishing a judgment. Thus we have introduced an additional clause in the LF paradigm for encoding a logic, namely: *external evidence as monads*. This class of monads is very well-behaved and so we can simplify the equational theory of the system. But, in fact, all our metatheoretic results carry through also in the general case, where we deal with a generic monad using Moggi's let_T destructor, together with its equational theory. *I.e.* we have provided an extension of LF with monads.

In this paper we consider the verification of predicates in locks as purely *atomic actions*, *i.e.* each predicate *per se*. But of course predicates have a logical structure which can be reflected onto locks. *E.g.* we can consistently extend $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$ by assuming that locks commute, combine, and entail, *i.e.* that the following types are inhabited: $\mathcal{L}^{\mathcal{P}}_{x,\sigma}[\tau] \to \mathcal{L}^{\mathcal{Q}}_{x,\sigma}[\tau]$ if $\mathcal{P}(\Gamma \vdash_\Sigma x : \sigma) \to \mathcal{Q}(\Gamma \vdash_\Sigma x : \sigma)$, $\mathcal{L}^{\mathcal{P}}_{x,\sigma}[\mathcal{L}^{\mathcal{Q}}_{x,\sigma}[M]] \to \mathcal{L}^{\mathcal{P}\&\mathcal{Q}}_{x,\sigma}[M]$, and $\mathcal{L}^{\mathcal{P}}_{x,\sigma}[\mathcal{L}^{\mathcal{Q}}_{y,\tau}[M]] \to \mathcal{L}^{\mathcal{Q}}_{y,\tau}[\mathcal{L}^{\mathcal{P}}_{x,\sigma}[M]]$.

We encoded call-by-value λ-calculus with Plotkin's classical notion of value. But the encoding remains the same, apart from what is delegated to the lock, if we consider other notions of value *e.g. closed normal forms* only for K-redexes [16]. This illustrates how monads handle side-conditions uniformly.

The way we dealt with the promotion rule in Elementary Affine Logic hints at the way to deal with *Pattern Matching* in $\mathsf{L}^{\mathsf{a}\times}\mathsf{F}$, and hence opens up the road to embedding *Logic Programming* and *Term Rewriting Systems* in type theory.

Acknowlegments. The authors would like to express sincere thanks to anonymous referees for their useful comments and useful remarks.

References

1. H. Albert. *Traktat über kritische Vernunft.* J.C.B. Mohr (Paul Siebeck), Tübingen, 1991.
2. N. Alechina, M. Mendler, V. de Paiva, and E. Ritter. *Categorical and Kripke Semantics for Constructive S4 Modal Logic.* In Proc. *CSL'01*, pp. 292–307, v. 2142 *LNCS*, Springer Berlin Heidelberg, 2001.
3. P. Baillot, P. Coppola, and U. D. Lago. *Light logics and optimal reduction: Completeness and complexity.* In Proc. *LICS*, pp. 421–430. IEEE Computer Society, 2007.

4. H. Barendregt. *Lambda Calculus: its Syntax and Semantics*. North Holland, 1984.
5. H. Barendregt and E. Barendsen. *Autarkic computations in formal proofs*. *Journal of Automated Reasoning*, 28:321–336, 2002.
6. G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. *Pure Pattern Type Systems*. In Proc. *POPL'03*, pp. 250–261. The ACM Press, 2003.
7. L. Carroll. *What the Tortoise Said to Achilles*. *Mind*, 4:278–280, 1895.
8. D. Cousineau and G. Dowek. *Embedding pure type systems in the lambda-Pi-calculus modulo*. In Proc. *TLCA*, v. 4583 of *LNCS*, pp. 102–117. Springer-Verlag, 2007.
9. M. Fairtlough and M. Mendler. *Propositional lax logic*. *Information and Computation*, 137(1):1–33, 1997.
10. M. Fairtlough, M. Mendler, and X. Cheng. *Abstraction and refinement in higher-order logic*. In Proc. *TPHOL'01*, number 2152 in *LNCS*. Springer Berlin Heidelberg, 2001
11. M. Fairtlough, M. Mendler and M. Walton. *First-order Lax Logic as a Framework for Constraint Logic Programming*. Tech. Rep., University of Passau, 1997.
12. D. Garg and M. C. Tschantz. *From indexed lax logic to intuitionistic logic*. Tech. Rep., DTIC Document, 2008.
13. R. Harper, F. Honsell and G. Plotkin. *A framework for defining logics*. *Journal of the ACM*, v. 40(1), pp. 143–184, Jan. 1993.
14. D. Hirschkoff. *Bisimulation proofs for the π-calculus in the Calculus of Constructions*. In Proc. *TPHOL'97*, number 1275 in *LNCS*. Springer-Verlag, 1997.
15. F. Honsell. *25 years of formal proof cultures: Some problems, some philosophy, bright future*. In Proc. *LFMTP'13*, pp. 37–42, New York, NY, USA, 2013. ACM.
16. F. Honsell and M. Lenisa. *Semantical analysis of perpetual strategies in λ-calculus*. *Theoretical Computer Science*, 212(1):183–209, 1999.
17. F. Honsell, M. Lenisa, and L. Liquori. *A Framework for Defining Logical Frameworks*. *v. in Honor of G. Plotkin, ENTCS*, 172:399–436, 2007.
18. F. Honsell, M. Lenisa, L. Liquori, P. Maksimovic, and I. Scagnetto. *An Open Logical Framework*. *Journal of Logic and Computation*, Oct. 2013.
19. F. Honsell, M. Lenisa, L. Liquori, and I. Scagnetto. *A Conditional Logical Framework*. In Proc. *LPAR'08*, v. 5330 of *LNCS*, pp. 143–157. Springer-Verlag, 2008.
20. F. Honsell, M. Miculan, and I. Scagnetto. *π-calculus in (Co)Inductive Type Theories*. *Theoretical Computer Science*, 253(2):239–285, 2001.
21. M. Mendler. *Constrained proofs: A logic for dealing with behavioral constraints in formal hardware verification*. In Proc. *Designing Correct Circuits*, pp. 1–28. Springer-Verlag, 1991.
22. E. Moggi. *The partial lambda calculus*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics, 1988.
23. A. Nanevski, F. Pfenning, and B. Pientka. *Contextual Modal Type Theory*. *ACM Transactions on Computational Logic*, 9(3), 2008.
24. F. Pfenning and C. Schürmann. *System description: Twelf – a meta-logical framework for deductive systems*. In Proc. *CADE*, v. 1632 of *LNCS*, pp. 202–206. Springer-Verlag, 1999.
25. B. Pientka and J. Dunfield. *Programming with proofs and explicit contexts*. In Proc. *PPDP'08*, pp. 163–173. ACM, 2008.
26. B. Pientka and J. Dunfield. *Beluga: A framework for programming and reasoning with deductive systems (system description)*. *Automated Reasoning*, v. 6173 of *LNCS*, pp. 15–21. Springer-Verlag, 2010.
27. K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. *A Concurrent Logical Framework I: Judgments and Properties*. Tech. Rep. CMU-CS-02-101, CMU, 2002.